3-D Model Building for Computer Vision[1]

B. Bhanu, C.C. Ho and T. Henderson
UUCS 85-112
September 30, 1985

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

This paper presents a Computer-Aided Geometric Design (CAGD) based approach for building 3-D models which can be used for the recognition of 3-D objects for industrial machine vision applications. The objects are designed using the Alpha_1 CAGD system developed at the University of Utah. A new method is given which uses the CAGD design and allows the points on the surface of the object to be sampled at the desired resolution, thus allowing the construction of multiresolution 3-D models. The resulting data structure of points includes coordinates of the points in 3-D space, surface normals and information about the neighboring points.

Index Terms: 3-D Model Building, 3-D Object Recognition, B-Splines, Computer-Aided Design, Computer-Aided Geometric Design, Multiresolution 3-D Models, Region Filling

Name and Present Address of the Corresponding author:

Bir Bhanu, Department of Computer Science, 3160 Merrill Engg. Building, University of Utah, Salt Lake City, Utah 84112

---

[1]This work was supported in part by NSF Grants DCR-8506393 and DMC-8502115

1. Introduction

The emergence of CIM (Computer Integrated Manufacturing) technology as a driving force in manufacturing engineering has provided opportunities and challenges to use geometric and functional models of real-world 3-D objects for the task of visual recognition and manipulation of these objects by robots [1]. CIM technology provides the database of objects as a byproduct of the design process. It allows the model-based recognition of 3-D objects to be simulated even before these objects are physically created. In this paper we present our ongoing work in defining how these designs could be used or modified in novel ways so as to be suitable for the task of recognition and manipulation.

The final CAGD model provided by the Alpha_1 system [2, 3, 4, 6] contains either surface patches or subdivided polygons or both. This is because after set operations are used to combine different parts of a complicated object [6] some surface parts may become extremely high order. The next step is to generate the vision models from the CAGD model. There are several approaches:

(a) **Universal Representation of Objects by Surface Points**: These can be either uniformly or nonuniformly sampled [1]. The procedure to get the nonuniformly sampled points is to simply subdivide the CAGD model until all polygons are smaller than some resolution. Then the center points of all the polygons can be obtained. The uniformly sampled points can be obtained by first filling line segments in the CAGD model and then using interpolation methods. Here information regarding surface normals and adjacency can also be obtained. This representation is the lowest level one. However, many higher level feature can be extracted from it by using the same procedures as used on the actual range data.

(b) **Intrinsic Surface Characteristics Defined by Points of High Curvature**: Since our CAGD model is based on surfaces, surface normal and surface curvature information can be easily obtained. Surface curvatures can also be computed by surface points obtained in (a).

(c) **Surface Representation by Edges/Arcs**: Edges and arcs can be extracted as boundaries of all surface patches used in the design process. The problem with this method is that the result is not unique since we can use different set of patches to model one object and get the same result in surface point representation. Edges can be obtained more accurately by using the surface curvatures.

(d) **Higher Level Surface Representation**: Methods (a) and (b) produce surface description in terms of the points on the surface of the object. To obtain a higher level surface representation we can use this information (adjacency, surface normals, etc.) in a more intelligent fashion if we use the polygons and surfaces obtained from the CAGD model directly. Here techniques like region growing can be effectively used.

Presently we are working on the above approaches for building 3-D models. In this paper we present one approach to model building using the Alpha_1 CAGD system. A new method is also given which uses the CAGD design and allows the points on the surface of object to be sampled at the desired resolution, thus allowing the construction of multiresolution 3-D models. The resulting data structure of points includes coordinates of the points in 3-D space, surface normals and information about the neighboring points. This information in turn can be used in building higher level description of the model.

2. Model Building Using the Alpha_1 CAGD System

The current modeling environment is the Alpha_1 Computer-Aided Geometric Design (CAGD) system. It models the geometry of solid objects by representing their boundaries as discrete B-splines. It allows the combination of simple objects into a more complex object using set operations. It supports several modeling paradigms, including direct manipulation of the B-spline surface, creation and combination of primitive shapes, and high-level shape operators such as bend, twist, warp and sweep. By using set operations on sculptured surfaces, the modeling task becomes easy and complete.

Steps to build a CAGD model: Here are some guide lines to build a CAGD model using Alpha_1.

1. Analyze the object - Usually a complex object can be divided into simpler parts which can be designed easily. A rule of thumb is "Use the same procedure that people use to create this object." Once the procedure is decided, one can concentrate on each subpart.

2. Measure Parameters - Make a precise measurement of parameters. If the design data is available, since it is best to use it. Some parameters may not be measured correctly, e.g., angles, because this requires special equipment. One way to solve this kind of problems is to use a computed value which is derived from another measured value. The other way is to specify another set of parameters. For example, an arc can be specified by its center point and two end points. It can also be specified by two tangent lines and its radius, or just three tangent lines. No matter how a model is designed, always use the most precise data.

3. When the surface patches of each part are designed, make sure that they have the correct orientation and set the adjacency information correctly. Otherwise an invalid object may have been created and the combiner will be unable to manipulate it.

4. The last step is to put all parts in the correct position and orientation and then use the combiner to perform the appropriate set operation on them.

3. Surface Point Extraction

As mentioned above, the final CAGD model consists of either surfaces or polygons or both. It is easy to convert any surface into polygons by repeatedly subdividing its control polygon but it is hard to reverse this procedure. Our strategy is to convert all the surfaces into polygons so as to make the problem uniform and easier. Unfortunately, the polygons we have are not all convex. One solution is to divide these concave polygons into a set of convex polygons. But it may not be computationally economical. Moreover, such a decomposition may not be unique. Another solution is to use a contour filling algorithm which can deal with 3-D concave polygons. We can fill line segments inside the polygon and then extract points on these lines.

There are several algorithms described by Pavlidis [5]. But none of them are suitable for our situation. In our case the vertexes can appear not only on some definite position, (such as a pixel in an image), but anywhere in space. Also we do not want to do any preprocessing of the contour, since there are a lot of polygons in the model and we may just fill a few discrete lines in one polygon. The algorithm is stated below. Its time complexity is linear.

3.1. Algorithm For Contour Filling

The main process of contour filling is to find the segments of an intersection line and the polygon. It can be done in two steps. First find all intersection points (junction points) and then decide which segments are inside the polygon.

The intersection points can be divided into three classes: the start point, end point and middle point of a line segment. Suppose we travel the line from left to right, the start point can be defined as the point whose left neighborhood is outside the polygon. The end point is the one whose right neighborhood is outside the polygon. And a middle

point is the one for which both left and right neighborhood are all inside the polygon. In Figure 1, point A is the start point, point B and C are middle points and point D is an end point. Point E is both start and end point. Now all we have to do is to divide all the intersection points into these three classes. Then skip all middle points and label the start points and end points in order alternatively.

To determine what kind a point is, we need topological information of the intersection points. It can be explained easily in two dimensional case. In Figure 1, the plane is divided by the line (this line need not be horizontal in the general case) into two regions, I and II. The direction of the contour is clockwise and there are five intersection points as shown in the figure. We find the directions of an edge when it passes through the intersection point. There are two directions for each point, in and out. For the edge going from region II to region I we mark the direction as 1 and if it goes from region I to region II we mark it as -1. And if the edge is lying on the line, we mark it as 0. For example, point A will be marked as (1,1) and point C is (0,1) point E is (1,-1). The direction information can be found easily at the same time when we find the intersection points. There are nine combinations of (in, out) directions. But they can be classified into four kinds only: flat(1), tangent(2), cut(2) and flat-cut(4) as shown in Figure 2. (The number in the parentheses is the number of combinations of (in,out) directions in a class.)

```
Flat point, in = out = 0
Tangent point, in = -out <> 0
Cut point, in = out <> 0
Flat-Cut point, Only one of in/out direction is 0.
```

Now the problem becomes: map these four kinds of intersection points into the three classes described above. Then we just skip all middle points and dump the start points and the end points alternatively.

Obviously, the flat point must be a middle point and the tangent point must be an end point if it is a start point or it may be a middle point if there is already a start point found for this segment. The cut point can be either a start or an end point. All these three kinds of points can be mapped easily. But to map the flat-cut point is not trivial, because it can be either a start, an end or a middle point. There is an important property of the edge directions. That is the out direction of an end point must be opposite to the in direction of its start point. As an example in Figure 1, A is the start point and D the end point. The in direction of A is 1 and the out direction of D is -1, but the out directions of point B and C are not -1. Our procedure will be:

Step 1: Sort the list of intersection points in order.
Step 2: Trace the list from the beginning to end.
Step 3: If this point is the first one or its previous point is an end point, then mark it as a start point. Otherwise do step 4.
Step 4: Test the out direction of this point. If it is opposite to the in direction of the previous start point then mark it as an end point. Otherwise it is a middle point.

But the above procedure may be incorrect if the contour is in the opposite direction of the cutting line. As shown in Figure 3, the direction of contour between D and C is from right to left. And the end point should be D instead of C. This condition will occur only in the tangent edge, i.e., the flat-cut point. This kind of error can be avoided by exchanging the in and out directions carefully. Thus, there is a restriction on the in/out direction and we can use it when a direction correction is required. For any two adjacent points, if the out direction of the first one is 0, on the line, the in direction of the second must be also 0 and if the out of first point is not 0 then the in of second point should not be 0. Of course, the in direction of a start point should not be 0 either. In Figure 3, there is a conflict between B and C, the out of B is not 0 and the in of C is 0. We have to exchange the in/out direction of point C to (-1, 0). And then we will change the in/out

direction of point D to (0,−1), since the conflict occurs on D after we change point C. After these corrections we can use the same procedure to find that D is the end point.

The final algorithm is:

Note:

Here we discuss the two-dimensional case and the cutting line is a horizontal line. In the next section we show how it can be easily extended to three dimension.

Inputs:

List of intersection points, each point contains both geometric and topological information.

Geometric info.:	point coordinates, (x,y).
Topological info.:	Traversing along the contour, we code the direction of an edge (in and out), on each intersection point.

Outputs:

A list of paired points. Each pair means the start and end of an interior line segment. Of course, some pair may be just a repetition of a tangent point.

Assumptions:

1. The contour is closed.
2. It does not intersects itself.
3. There is no hole inside the contour.

Step 1: Sort the intersection points listed by x coordinate (left-to-right)

Step 2: Trace the sorted list from left-to-right and for each point do

```
    If the point is the first one or
    the previous one is an end point
    then
        do step 3
    else
        do step 4
```

Step 3: Output this point { start point }

```
If it is a tangent point { in = -out }
then
    output it again { end point }
else
begin
    if the in direction is 0
        then exchange the in and out direction of this
         point.
    { the in direction of a start point will not be
      0 after the adjustment }
    save the in direction of this start point.
end
```

```
Step 4: If it is either a flat point or a tangent point
        then
            skip it { middle point }
        else
            do step 5

Step 5: If there is a conflict regarding the restriction
        as described in the above, then exchange the in and
        out direction of this point.
        If the out direction of this point is opposite to the
        in direction of the previous start point after the
        adjustment, if required
        then
            output this point { end point }
        else
            skip it { middle point }
```

Figure 4 shows a two dimensional example. Figure 4(a) is the input contour. Each cross represents a vertex of the contour in the output of the combiner (which allows the use of a set of Boolean operators to combine sub parts) of Alpha_1. Some of these vertexes which appear to be redundant are also part of the other polygons. When considering a polygon they can be easily removed by finding the angle between the two neighboring vertexes of a vertex under consideration. The direction of this contour is not significant for the use of the above algorithm. Figure 4(b) is the result of the above algorithm applied to Figure 4(a). Finally, in Figure 4(c), we extract all the surface points from the line segments by a user defined resolution

The complexity of this algorithm is O(n + m*log(m)), where n is number of vertexes of the input contour and m is the number of intersection points of one cutting line. Usually the number of intersection points is much less then the number of vertexes, and the complexity is linear. In the worst case, m is equal to (n-1) and the complexity becomes n*log(n). This is due to the sorting of the intersection points.

4. An Example

Now we will show how to follow the procedure described above to design the object in Figure 5. To design the CAGD model in the Alpha_1 system, we can build the complete object in a stepwise manner. First, we design the plate in Figure 6(a) and the outline cutter surface in Figure 6(b). By cutting the plate with this outline surface we can get the turtle like shape. Then we design the dent part, scratches and all the holes in Figure 6(c) to 6(g). To design these parts we design curves using B-splines first and then use various high level operators for surface construction, such as revolving a curve about an axis, extruding a curve in some direction and filling the surface between two curves. Figure 6(i) shows the position of all these small parts. At this moment the model is almost done except the threads. There are seven thread in this object, the center, the head and the five small ones. Threads can be designed by filling two surfaces between a twisted curve and the different part of another twisted curve. Figure 6(h) is the center one, the others are the same except for their radius and pitch. Figure 9 is a close view of the center portion. It can be seen that the lower scratch is different from the others. It is because they are different in the object.

Next we use the contour filling algorithm described before to find the surface points and their normals. First we convert parts of the representation which are still surfaces, in our CAGD model after the set operation, into polygons by subdivision [4]. Now we have three dimensional polygons only. The only problem to extend the algorithm as described in the last section to 3-D polygons, is to choose the direction to get the cutting lines and to choose points which are within the desired resolution. Of course, we cannot always cut in x-y direction otherwise we will get no point in the surface which is parallel to the z direction. The way we used here is to find the bounding box first and choose the two directions of this box in which there is maximum spread to be our cut direction for the

line and points, respectively. A bounding box of a polygon is specified by the minimum and maximum x, y, z coordinates of of all its vertexes. This allows us to guarantee that the distance between any two points must be smaller than the diagonal of the cube which has the increment value we use in cutting the polygon as its side. The length of this diagonal is the resolution, the maximum distance between any two adjacent points. Figure 7 shows the surface points with 0.2 inch resolution. In Figure 8, we show the point normals with 0.4 inch resolution. The output polygon of the combiner will give the normal of every vertex which is computed from the surface. When we interpolate the intersection points we can get their normals from the adjacent vertexes by linear interpolation. And we do the same thing when we extract the surface points from the interior line segments of the polygon.

References

[1] B. Bhanu and T. Henderson.
CAGD Based 3-D Vision.
In *International Conference on Robotics and Automation.* IEEE, March, 1985.

[2] E.S. Cobb.
Design of Sculptured Surfaces Using the B-spline Representation.
PhD thesis, University of Utah, June, 1984.

[3] E. Cohen.
Some Mathematical Tools for a Modeler's Workbench.
IEEE Computer Graphics & Applications :63-66, October, 1983.

[4] E. Cohen, T. Lyche and R.F. Riesenfeld.
Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics.
Computer Graphics and Image Processing 14(2):87-111, October, 1980.

[5] T. Pavlidis.
Algorithms for Graphics & Image Processing.
Computer Science Press, 1982.

[6] S.W. Thomas.
Modelling Volumes Bounded by B-Spline Surfaces.
PhD thesis, University of Utah, June, 1984.

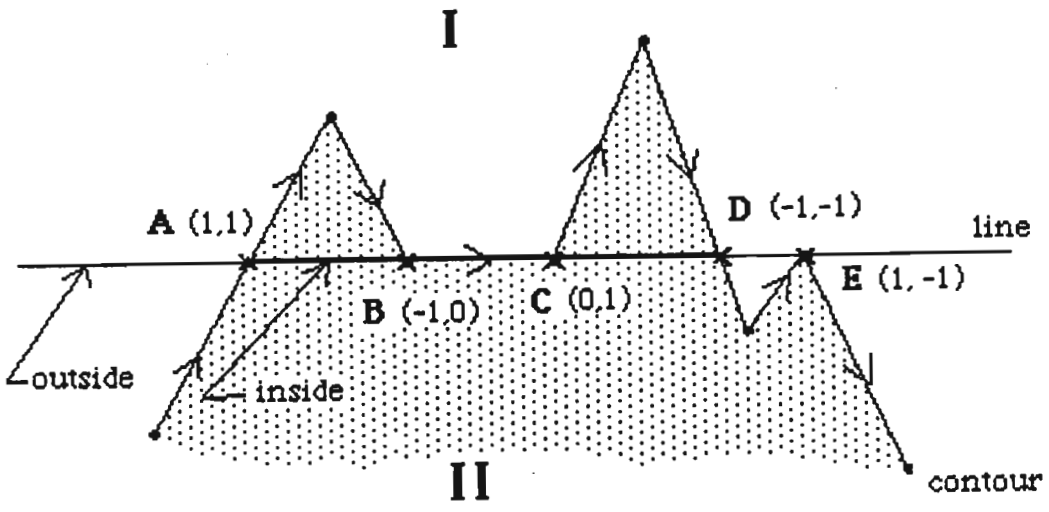
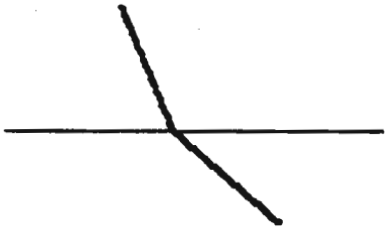


Fig. 1 Direction of edges

2(a) Flat point, in = out = 0

2(b) Tangent point, in = -out <> 0



2(c) Cut point, in = out <> 0

2(d) Flat-Cut point,
Only one of in/out direction is 0.

Fig. 2 Four kinds of junction points.

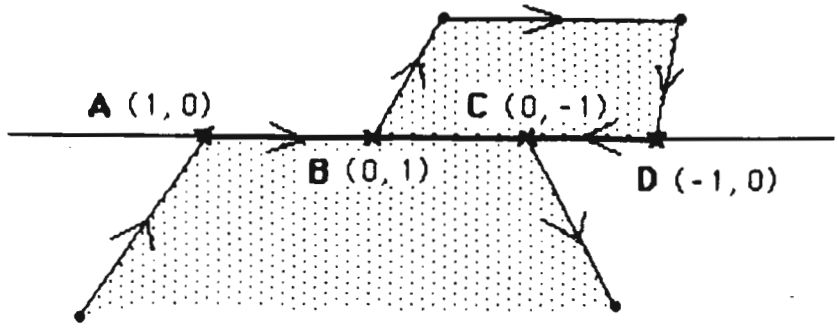
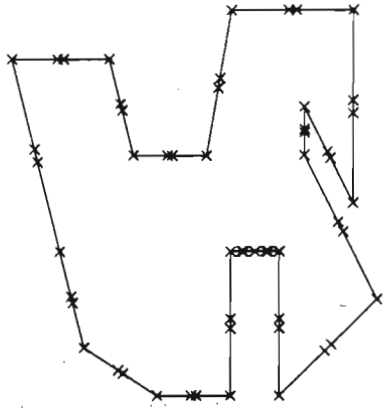
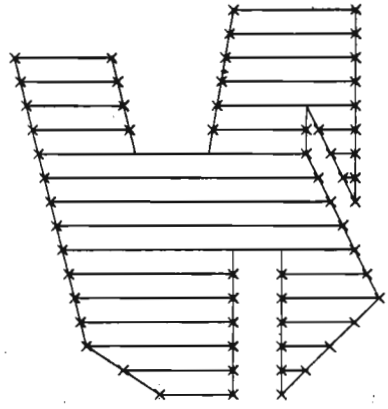


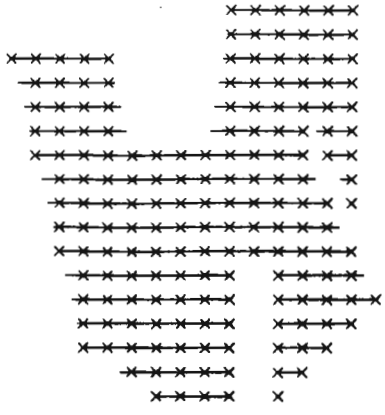
Fig. 3 The restriction on in/out directions.



4(a) Input contour



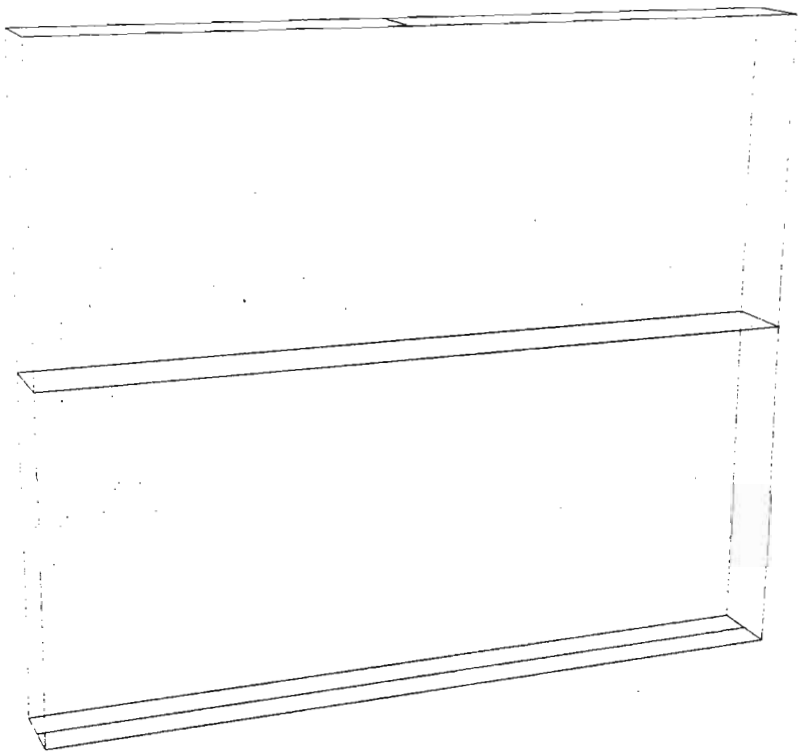
4(b) Interior line segments



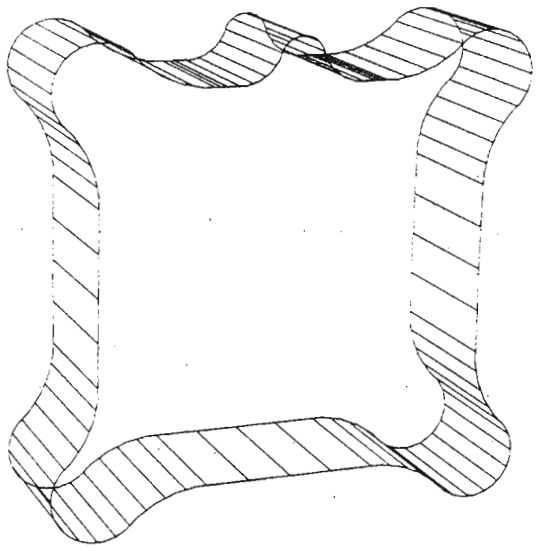
4(c) Surface points on the polygon

Fig. 4 A two dimentional example.

Fig. 5 A picture of the final CAGD model of the object

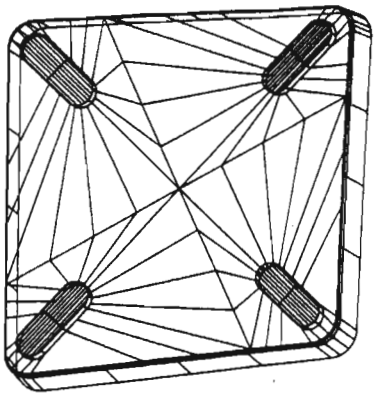


6(a) Plate

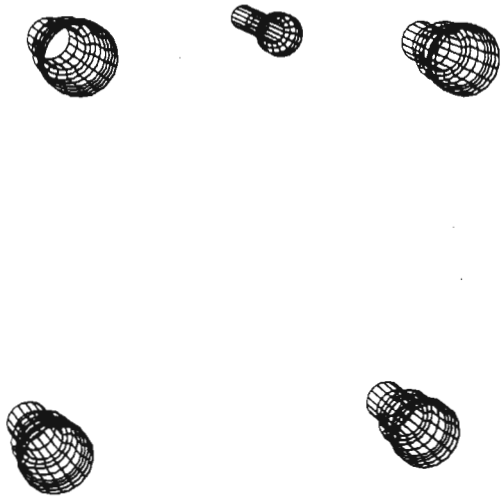


6(b) Outline cutter

Fig. 6 Subparts for designing the work piece

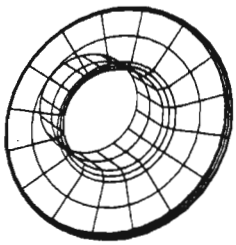


6(c) Outer dent part and the corner scratches



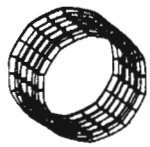
6(d) Head hole and corner holes

6(e) Four arc scratches

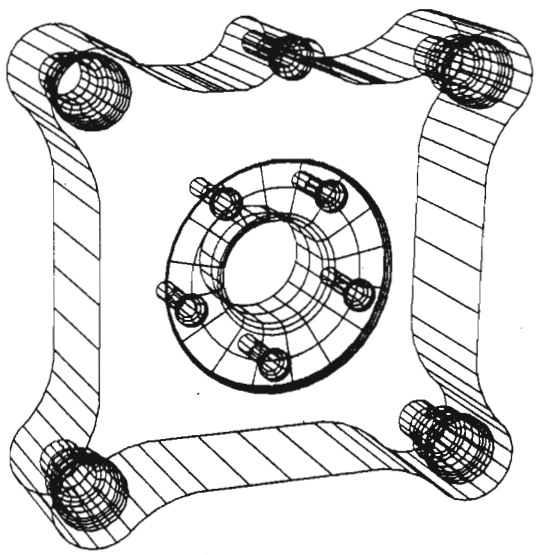


6(f) Inner dent part and the center hole

6(g) Five small holes



6(h) Center thread



6(i) The position of the subparts in the model of the work piece.

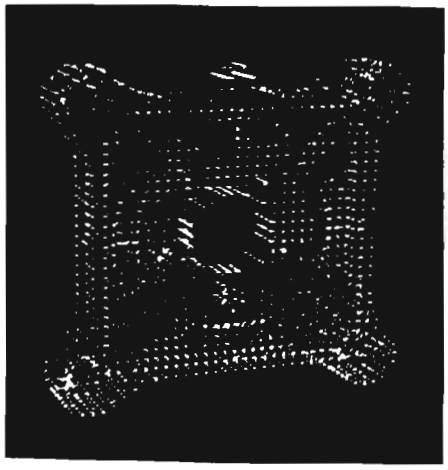


Fig. 7 Surface points extracted from the CAGD model, 0.2 inch spacing.

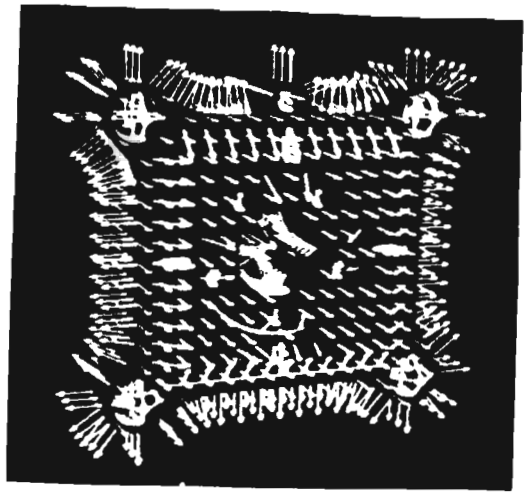


Fig. 8 Normals at the surface points, 0.4 inch spacing.

Fig. 9 A closer view of the center portion of the work piece.

List of figures

Fig. 1 Direction of edges - The direction of edge going from region II to region I is 1, from region I to region II is -1 and for the tangent edge it is 0. At the junction point of two edges, there are two directions, namely, in direction and out direction. For point A, it is (1,1) and for point C, it is (0,1). The arrows indicate the direction of the contour. The line need not be horizontal.

Fig. 2 Four kinds of junction points. By using the in/out direction at each junction point, there are 9 different combinations of 4 kinds: (a) flat, (b) tangent, (c) cut, (d) flat-cut. A flat point is the one with both its in and out directions characterized as 0,i.e., it is (0,0). There are two possiblities for tangent point, (1,-1) (-1,1); two for cut point, (1,1) (-1,-1); and four for flat-cut point, (0,1), (0,-1), (1,0), (-1,0).

2(a) Flat point, in = out = 0
2(b) Tangent point, in = -out <> 0
2(c) Cut point, in = out <> 0
2(d) Flat-Cut point, Only one of in/out direction is 0.

Fig. 3 The restriction on in/out directions. In order to get the correct end point of a line segment, we have to exchange the in/out directions of flat-cut point if the direction of the contour at this point is opposite to the line. (The direction of line is always from left to right.) It can be detected by the restriction on in/out directions. If the out direction of a previous point is not 0 then the in direction at this point should not be 0 otherwise the contour is in opposite direction at this point and its in and out directions have to be interchanged. We have to first change C to (-1,0) and then D to (0,-1).

Fig. 4 A two dimentional example. Fig. (a) shows the input contour. The points marked as x's are the vertexes of this contour. Fig. (b) shows all the interior line segments. The required points are the start and end points of these segments. Fig. (c) shows all the surface points of this polygon. They are extracted from the line segments in fig. (b).

4(a) Input contour
4(b) Interior line segments
4(c) Surface points on the polygon

Fig. 5 A picture of the final CAGD model of the object

Fig. 6 Subparts for designing the work piece

6(a) Plate
6(b) Outline cutter
6(c) Outer dent part and the corner scratches
6(d) Head hole and corner holes
6(e) Four arc scratches
6(f) Inner dent part and the center hole
6(g) Five small holes
6(h) Center thread
6(i) The position of the subparts in the model of the work piece.

Fig. 7 Surface points extracted from the CAGD model, 0.2 inch spacing.

Fig. 8 Normals at the surface points, 0.4 inch spacing.

Fig. 9 A closer view of the center portion of the work piece.